Universidad Zaragoza
1542

Trabajo Fin de Grado

Habilitación del control remoto y visualización para el modelo de un sistema automatizado de almacenamiento y recuperación.

Autor

Adrián Marco Artigas

Director

Orthodoxos Kipouridis

Ponente

Antonio Romeo Tello

Universidad de Zaragoza

Año 2014

RESUMEN

Con el fin de hacer frente a la creciente necesidad de flexibilidad, los sistemas de flujos de materiales de hoy en día se mueven hacia enfoques descentralizados y modulares. Sin embargo, el funcionamiento y el control de la mayoría de estos sistemas están basados en estándares específicos del fabricante, lo que resulta en la pérdida de su reutilización.

Dentro de este enfoque tiene lugar un gran proyecto de colaboración entre universidades alemanas, en el que se pretende implementar un sistema de almacenamiento y recuperación de materiales totalmente automatizado y controlado por SAP. Este trabajo fin de grado es una pieza de ese gran proyecto, con el se habilitará la visualización del sistema a través de un escenario tridimensional.

El trabajo comienza con el modelado 3D del sistema haciendo uso de herramientas CAD. Para dar movimiento a ese escenario se hará uso de KoDeMat, un software en desarrollo que utiliza el motor de juegos de JMonkey Engine. Basada en agentes Java ya existentes, se diseñará una API que permita la comunicación entre KoDeMat y Simulink. Por último, se hará una simulación del sistema y quedará listo para utilizarse en el modelo real.

INDICE

INDICE DE ILUSTRACIONES

INDICE DE TABLAS

1 Introducción

El presente documento recoge el trabajo realizado en el departamento de logísitca y manejo de materiales, Lehrstuhl Fördertechnik Materialfluss Logistik, de la Universidad Técnica de Múnich, TUM.

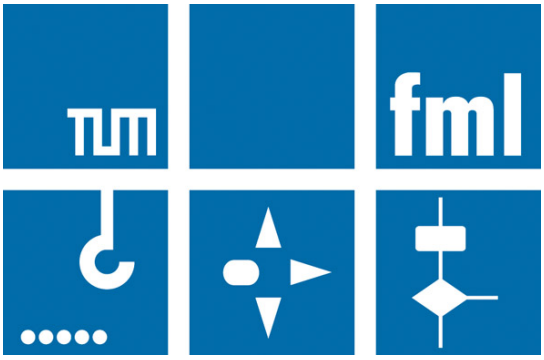

Fig. 1. Logo del departamento de logística en TUM

El formar parte de este proyecto ha sido posible gracias a una beca de colaboración que la Universidad Técnica de Múnich ofertó a través de IAESTE, una asociación internacional que favorece el intercambio de estudiantes para realizar prácticas en el extranjero.

Para una mejor comprensión de lo descrito en el documento es necesario tener conocimientos básicos de programación orientada a objetos y Matlab.

1.1 Contexto

En la última década, la investigación aplicada en el campo de los sistemas automatizados de flujo de materiales ha introducido enfoques de control descentralizados con numerosas aplicaciones comerciales ya abriéndose camino en el mercado. La razón principal para este desarrollo es la demanda de sistemas flexibles y reutilizables, que pueden adaptarse a los requisitos cambiantes en ámbito logístico en las instalaciones de hoy en día. Tal paradigma de control descentralizado se define como "Internet of Things" (IoT) para instalaciones logísticas y que describe un enfoque descentralizado para el flujo de materiales. IoT permite la separación del control operacional y de la máquina habilitando el desarrollo de sistemas altamente modulares.

Hoy en día, nuevos avances tecnológicos, tales como arquitecturas software orientadas a servicios y distribución de datos, ofrecen nuevas posibilidades a los fabricantes de sistemas, integradores de sistemas e investigadores, para colaborar en proyectos comunes mediante el aprovechamiento de los beneficios que la descentralización ofrece.

1.2 Objetivos y alcance del proyecto

El trabajo realizado forma parte de un gran proyecto, orientado a la descentralización y modularización de un sistema de almacenamiento y recuperación de materiales. El sistema con el que se trabaja es en realidad una maqueta, sin embargo bien podría tratarse de un sistema real.

Se parte de un sistema descentralizado ya desarrollado con base en servicios SAP. A través de estos servicios se pueden dar órdenes de almacenamiento de pedidos, procesado y recuperación. El modelo cuenta con una computadora PLC, que es capaz de comunicarse con un ordenador de control a través de mensajes SOAP. Estos fueron proyectos desarrollados por otros estudiantes.

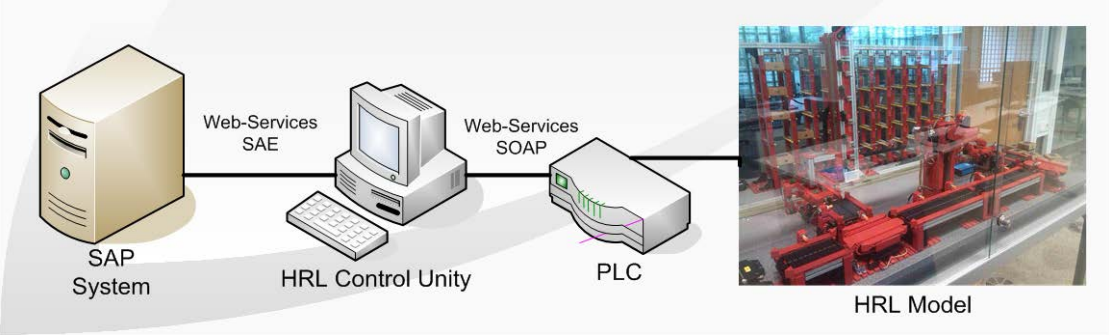

Fig. 2. Diagrama de conexión entre el modelo de almacén y los diferentes elementos de control

El objetivo del presente proyecto se centra en la habilitación de la visualización del sistema de almacenamiento y recuperación. Para ello se cuenta con clientes Java ya desarrollados, en particular la aplicación KoDeMat, que se explica en la siguiente sección.

Para empezar, será necesario modelar tridimensionalmente los elementos que componen la maqueta, utilizando herramientas CAD de diseño 3D y poder construir con ellas un escenario. La construcción del escenario se hace con una herramienta software llamada JMonkey Engine, utilizada por KoDeMat. JMonkey dispone de un motor gráfico y servirá para dar movimiento al escenario. Se pretende utilizar la herramienta Matlab para poder leer el estado de los sensores del sistema, por lo que se necesitará establecer un punto de comunicación entre Matlab y el cliente de visualización. Esto se llevará a cabo con la implementación de una API para este sistema automatizado.

Con todo ello, el escenario 3D cobrará vida y será posible visualizarlo desde cualquier PC que utilice la aplicación KoDeMat y que, a su vez, se encuentre conectado a la red virtual VPN específica para este propósito.

1.3 Organización del documento

Este documento está estructurado de la siguiente manera:

Primero, se presenta cual ha sido el punto de partida, con ello se da a conocer los elementos base utilizados para el desarrollo del proyecto.

Se comentarán las herramientas software utilizadas junto a una breve descripción y también las fuentes de consulta ó referencias.

Una vez conocidas las herramientas con las que se cuenta, el documento se centra en el trabajo realizado por el estudiante. Este trabajo queda bien diferenciado en tres etapas: el modelado 3D de los componentes del modelo real, el desarrollo de código Java para la implementación de una API que sirva para comunicar Matlab con el cliente de visualización y la fase de simulación.

2 Punto de partida

A continuación se dan a conocer los elementos base del proyecto sobre los que se ha trabajado. Estos son la maqueta del sistema logístico de almacenamiento y recuperación y la arquitectura software basada en agentes java.

2.1 El sistema de almacenamiento y recuperación

El modelo de almacén sobre el que se ha trabajado consiste en un maqueta a pequeña escala de un posible sistema real. Esta maqueta ha sido montada utilizando piezas Fischertechnik, sensores capacitivos y motores eléctricos. Utiliza un PLC (modelo CP1131 de CANTROL) para controlar el estado de los sensores y los actuadores. El almacén es designado dentro del proyecto como *HRL* (en alemán hochregallager), que significa almacén de gran altura.

El modelo consta de dos sistemas principales, el stock real con la unidad de almacenamiento y recuperación y las plataformas de transporte. La Fig. 3. Esquema del sistema de almacenamiento y recuperación, ilustra la planta del modelo. A modo resumen se describe, a continuación, el funcionamiento básico del sistema y se expone a modo tabla los diferentes elementos y su funcionalidad. No se va a entrar en detalle con la lógica de funcionamiento del sistema, ya que su implementación, ya desarrollada, no ha sido el objetivo de este proyecto.

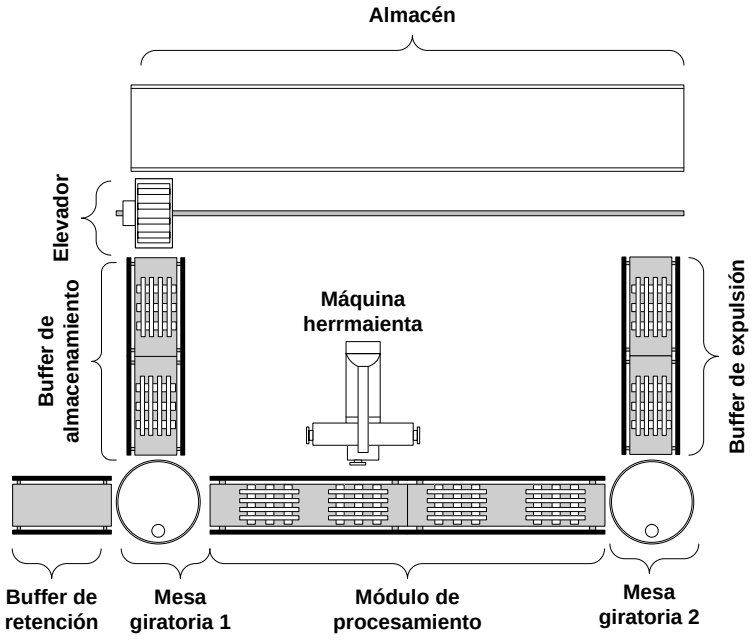

Fig. 3. Esquema del sistema de almacenamiento y recuperación

El proceso comenzaría con la recepción de los componentes semiacabados a través de la mesa giratoria dos. Los componentes son arrastrados a través del módulo de procesamiento hasta la máquina de procesado y se trasladan al buffer de almacenamiento pasando por la mesa giratoria uno. Cuando el elevador no es lo suficientemente rápido colocando los componentes en las celdas del almacén, se origina un cuello de botella en el buffer de almacenamiento. Ahí entra en juego el buffer de retención, éste almacenará los pedidos temporalmente para no interrumpir la línea de procesado. Cuando el buffer de

almacenamiento esté libre, se procederá a trasladar los pedidos que hayan quedado en el buffer de retención. Finalmente, cuando se quiere retirar un componente del almacén se hace a través del buffer de expulsión y de la mesa giratoria dos.

Módulo	Descripción
Mesa giratoria 2	Mesa giratoria que sirve como punto de entrada y salida de cajas en el almacén HRL.
Módulo de procesamiento	Traslada los semiacabados hacia la máquina herramienta para el procesado y hacia la mesa giratoria para su almacenamiento.
Mesa giratoria 1	Mesa giratoria que comunica la línea de almacenamiento y la línea de retención.
Buffer de retención	Almacena temporalmente los pedidos cuando el buffer de almacenamiento no avanza lo suficientemente rápido.
Buffer de almacenamiento	Transporta los pedidos procesados hacia el elevador para su almacenamiento.
Elevador	Almacena y recupera los pedidos del almacén. Puede ocasionar un cuello de botella en el buffer de almacenamiento.
Almacén	Almacena los pedidos procesados en una de las cincuenta celdas disponibles.
Buffer de expulsión	Recibe los pedidos almacenados y los transporta hacia la mesa giratoria ó giradiscos 2 para su expulsión.
Máquina herramienta	Realiza una operación en los semiacabados. Concretamente se trata de una máquina de fresado.

Table 1. Resumen de los diferentes componentes del sistema según su funcionalidad

En los anexos puede encontrarse una lista de los diferentes componentes, con la nomenclatura utilizada en este proyecto y una imagen de cada uno, extraídas durante la fase de modelado 3D.

2.2 Clientes KoDeMat y Hazelcast

KoDeMat es una plataforma de colaboración que tiene como objetivo ser utilizada durante la fase de diseño de un sistema logístico, donde diversos subsistemas deben configurarse para ser compatibles entre sí. Más adelante, en una fase de pruebas, podría utilizarse para testear y visualizar el funcionamiento del sistema. No obstante, KoDeMat no ha de entenderse como una herramienta de sustitución del software de simulación del respectivo subsistema.

Los requisitos previos incluyen que los socios implicados en el proyecto estén interconectados a través de una red privada virtual (VPN) y que cada socio tenga acceso al software de simulación basado en agentes de su respectivo sistema de logística, de esto se encarga el cliente Hazelcast.

Hazelcast es un Data Grid en Java, o dicho de otra forma una plataforma escalable para la distribución de datos.

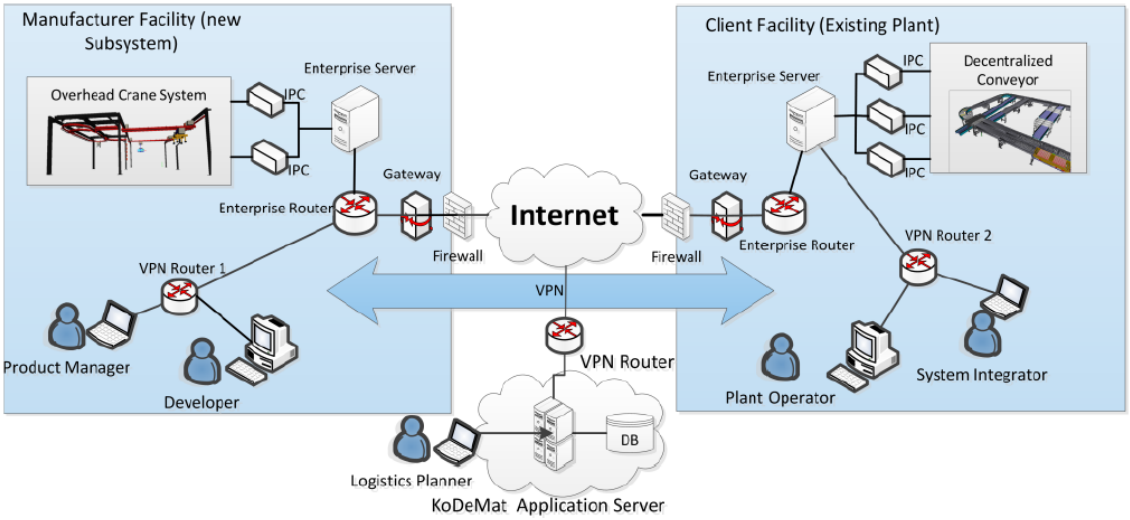

Fig. 4. Diagrama de un escenario ejemplo para la plataforma KoDeMat.

La imagen muestra un diagrama ejemplo de trabajo colaborativo utilizando la aplicación. Diversas personas trabajan juntas en el proyecto sin tener necesidad de estar en el mismo lugar.

La plataforma KoDeMat junto a Hazelcast permite a los usuarios visualizar simultáneamente el estado de todos los sistemas conectados en un entorno virtual 3D. Los usuarios pueden interactuar unos con otros, así como con los modelos. Un ejemplo de dicha interacción podría ser el señalar una posición en un sistema indicando un posible punto muerto, o también la adición de un elemento visual que resalte una posición específica del modelo 3D para llamar la atención de otros usuarios ante un problema operativo potencial.

3 Herramientas utilizadas y fuentes de consulta

3.1 Sumario del software utilizado

- SketchUpMake v.14: software de diseño gráfico y modelado en tres dimensiones. Se utiliza frecuentemente en entornos de arquitectura, ingeniería civil, diseño industrial, diseño escénico, GIS, videojuegos o películas.

- Blender v.2.72: software de modelado, iluminación, renderizado, animación y creación de gráficos tridimensionales. Con él también es posible la composición digital y el desarrollo de vídeojuegos.

- JMonkey Engine SDK 3.0: software de código abierto, indicado especialmente para los desarrolladores de videojuegos utilizando el lenguaje Java. El software está destinado a la amplia accesibilidad y una rápida implementación.

- Matlab v.R2013b: es a la vez un lenguaje de alto nivel y un entorno interactivo para el cálculo numérico, visualización y programación. Matlab se utiliza para análisis de datos, el desarrollo de algoritmos y la creación de modelos y aplicaciones.

3.2 Referencias

1) Documento *Cloud-based Platform for Collaborative Design of Decentralized Controlled Material Flow Systems in Facility Logistics*. (2014). Technische Universität München.
2) MathWorks (2014). http://es.mathworks.com Accedida durante septiembre, octubre y noviembre de 2014.

4 Habilitación de la visualización

Habilitar la visualización del sistema de almacenamiento es una gran tarea que se divide en varias fases. En este apartado se explica todo el trabajo personal llevado a cabo en el proyecto, sus etapas han sido las siguientes: modelado 3D del sistema de almacenamiento y recuperación de materiales, aprendizaje sobre la relación entre Matlab y Java, desarrollo de una API en JMonkey que gestione la visualización, control de la API desde Simulink y simulación del sistema.

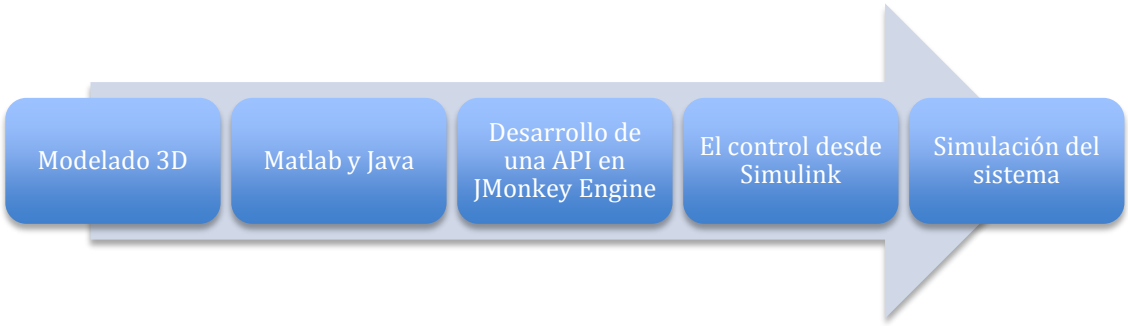

Fig. 5. Despliegue cronológico de las diferentes etapas seguidas en la realización del proyecto

El trabajo comienza con el modelado 3D del sistema haciendo uso de herramientas CAD. Con los modelos de los diferentes componentes del almacén podrá empezar la siguiente etapa.

Se necesitará disponer de los conocimientos necesarios sobre cómo hacer uso de los clientes Java de los que se dispone y de cómo poder utilizarlos desde Matlab. Se trata de una pequeña etapa de aprendizaje y experimentación, necesaria para saber cómo enfocar el desarrollo de una API que comunique ambas plataformas.

Una vez sabido cómo establecer esa comunicación, se pasa al desarrollo de la API. Para ello se utiliza el entorno de desarrollo integrado JMonkey Engine, en ella se desarrolla el código Java necesario para gestionar la visualización de los componentes. Ésta etapa y la siguiente, control desde Simulink, van de la mano, se ha trabajado en ambas de forma paralela desarrollándolas de forma conjunta, ya que la aplicación desarrollada en Simulink utilizará la API.

Para concluir, se pasa a la fase de pruebas. Se trata de modelar parte de la lógica del PLC y crear un pequeño circuito a recorrer por una o varias cajas. Con este modelo podrá testearse el sistema de visualización desarrollado sin necesidad de estar conectado al modelo real.

Con el sistema de visualización terminado y operativo será posible acoplarlo al PLC real. Bastaría con reemplazar el sistema modelado en Simulink por la lectura de las variables del sistema real a través del ordenador que esté conectado al PLC.

4.1 Modelado 3D

Hoy en día puede encontrarse una gran variedad de software destinado al dibujo 3D. El gran abanico de recursos puede hacer difícil la elección del más apropiado para la tarea a desarrollar.

La maqueta a dibujar está compuesta de pequeñas piezas Fischertechnik, es un sistema de montaje similar al de algunos juguetes utilizados por niños para entretenerse. Fischertechnik ofrece su propio software, *Fischertechnik Designer*, para construir virtualmente los modelos, permitiendo también testear el ensamblaje y funcionamiento de engranajes y demás partes móviles.

Lo más indicado, por tanto, sería utilizar el software oficial de Fischertechnik, pero debido a que no se disponía de la licencia y que la versión gratuita no permite trabajar con más de veinte piezas, finalmente se ha utilizado *SketchUp*.

SketchUp supone una buena alternativa, ya que permite importar multitud de librerías de componentes prediseñados y también componentes Fischertechnick. Además, pueden dibujarse geometrías de forma muy sencilla y que son de gran ayuda para complementar los modelos.

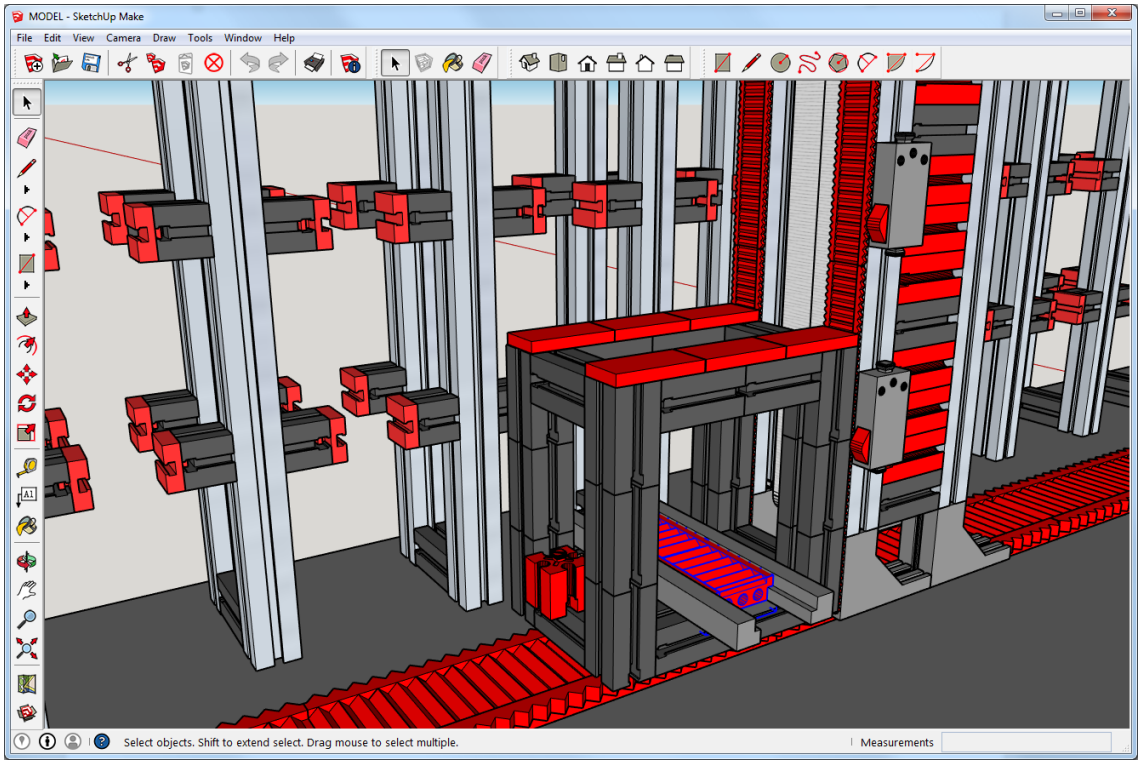

Fig. 6. Captura de SketchUp en la que se puede ver el elevador y la estantería.

No hay que olvidar que el objetivo del modelo 3D es incorporarlo a *Jmonkey Engine* y crear un escenario en el cual sea posible controlar los diferentes elementos.

JMonkey Engine necesita archivos con formato *blend*, formato utilizado por *Blender*, para poder importarlos y convertirlos a su propio tipo de archivo, el formato *j3o*. Esto es así ya que JMonkeyEngine y Blender son dos herramientas software complementarias en el desarrollo de videojuegos.

SketchUp permite la exportación de sus modelos en multitud de formatos, sin embargo el formato blend no está entre ellos. Por lo tanto, hay que encontrar un tipo de archivo común, es decir, uno al que SketchUp sea capaz de exportar y que su vez Blender pueda importar. Éste ha sido el formato *3ds*.

Blender no se ha utilizado únicamente como intermediario en la conversión de archivos. Otras tareas realizadas en él han sido la asignación de una referencia local dentro de cada componente y también la disminución del número de vértices en los modelos. Gracias a las referencias locales, resulta más sencillo desplazar los objetos "caja" a lo largo de los objetos "máquina" y con la reducción de vértices se consiguen unos tamaños de archivo más reducidos y un menor coste computacional en su representación.

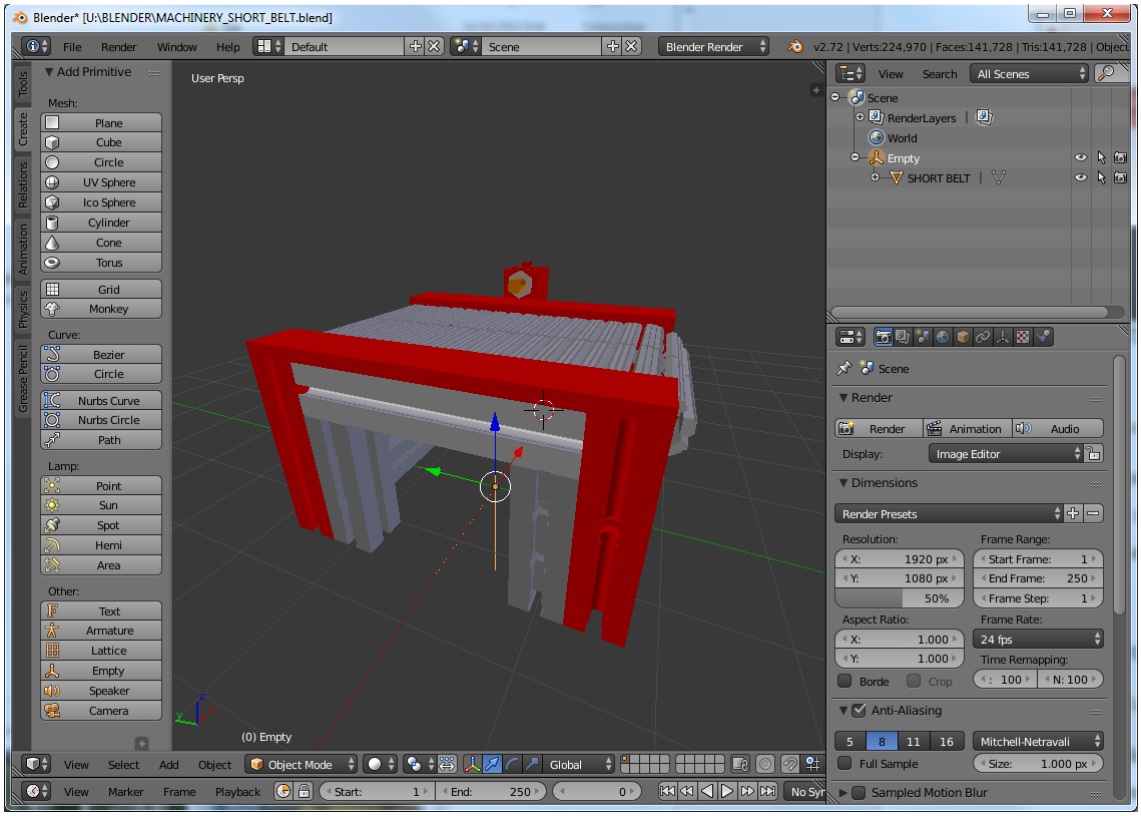

Fig. 7. Máquina transportadora en Blender con su referencia local.

Con la obtención de los archivos, uno para cada componente en formato blend, finaliza la tarea de modelado 3D. En los anexos del documento pueden consultarse la lista de componentes creados y sus fotografías.

4.2 Matlab y Java

La relación existente entre Matlab y Java es un punto clave en el proyecto. Por un lado, los agentes encargados de la visualización y de la comunicación en la red han sido desarrollados en lenguaje Java, por otro lado, interesa utilizar Matlab como interfaz entre los sensores del almacén y el agente de visualización.

Toda instalación de MATLAB incluye la máquina virtual de Java (JVM). Es posible usar el intérprete Java a través de comandos Matlab y se pueden crear y ejecutar programas, así como crear y acceder a objetos Java. Es esencial que la versión de la JVM de Matlab sea la misma que la utilizada por la IDE en la que se desarrolla el código Java, en este caso JMonkey Engine. La versión de la JVM utilizada ha sido la 7.

Para poder ejecutar el código Java desarrollado primero ha de ser incluido en la ruta de librerías Java que utiliza Matlab. El código previamente ha de ser compilado y construido resultando en un archivo comprimido especial *jar*.

La ruta de librerías de Matlab está dividida en dos categorías: la ruta estática y la ruta dinámica. Para añadir la ruta de forma estática ésta ha de incluirse al fichero *javaclasspath.txt* que utiliza Matlab. Para añadir la ruta de forma dinámica ha de hacerse mediante la función *javaaddpath* seguida de la ruta del fichero jar. En este caso:

- `javaaddpath U:\MonkeyEngine\KoDeMat_HRL\dist\KoDeMat_HRL.jar`

El uso del código Java se realiza mediante llamadas a métodos especificando la ruta en la que se encuentra dicho método dentro del fichero jar; es decir, la ruta de la clase en la que se encuentra el método. Los métodos de una clase objeto pueden ser: estáticos ó no estáticos. Un método estático podrá ser llamado directamente desde Matlab, mientras que para invocar métodos no estáticos será necesario crear previamente un objeto ó instancia en el espacio de trabajo de Matlab. Para crear un objeto Java se utiliza la función *javaObject*:

- `javaObj = javaObject('className', arg1, arg2, argN);`

Esta expresión crea un objeto de la clase "className" y guarda su dirección en la variable "javaObj". Los argumentos deben ajustarse a uno de los constructores de la clase.

Para invocar un método Java se puede utilizar la función *javaMethod* de las siguientes formas:

- `javaMethod('methodName', 'javaObj', arg1, arg2, argN);`

Esta expresión se utiliza cuando se ha creado previamente, un objeto en el espacio de trabajo.

- `javaObject('staticMethodName', 'className', arg1, arg2, argN);`

Esta expresión no requiere tener definido un objeto java, solamente podrán invocarse métodos estáticos.

En lo que concierne a este proyecto, se necesita una forma sencilla de ejecutar código desde una función en Simulink. Con las funciones anteriores podrán invocarse los métodos necesarios para gestionar la visualización del sistema de almacenamiento, enviando el estado de los sensores correspondientes.

4.3 Desarrollo de una API en JMonkey Engine

JMonkey Engine es un entorno de desarrollo integrado Java, basado en NetBeans, para desarrollo de videojuegos. JMonkey es capaz de importar en sus proyectos modelos 3D en formato blend e incorpora sus propias librerías para hacer uso del entorno gráfico. Con el uso de estas librerías es posible añadir y manejar los distintos elementos que componen un escenario de forma sencilla.

Este punto del proyecto empieza con la adición de los modelos 3D desarrollados en SketchUp y posteriormente convertidos por Blender a formato blend. El proceso es sencillo, basta con importar los modelos al proyecto y JMonkey los convierte automáticamente a formato j3o.

En JMonkey es posible construir un escenario compuesto de objetos no móviles y después añadir los componentes que, mediante código, serán manejables. En este proyecto, los objetos móviles son solamente el elevador, las mesas giratorias y las cajas que circularán a lo largo de las cintas transportadoras. No obstante, todos los objetos serán considerados cómo objetos móviles, ya que cuando una caja tenga que cambiar de máquina se hará convirtiendo el componente caja en hijo de un componente máquina. Esto significa, que la caja se moverá dentro de las coordenadas del componente padre. De este modo, en caso de cambiar la posición de una máquina, no será necesario definir nuevas coordenadas a las que mover la caja.

Hay que recordar que este proyecto utiliza el cliente de visualización KoDeMat, en él se dispone de una interfaz con métodos útiles para crear otros métodos más sencillos y generalistas de los cuales se han utilizado los siguientes:

- `public VisuComponent createModel(String modelName, String modelAssetPath)`

EL método “createModel” devuelve un componente de tipo "VisuComponent". Cuando un VisuComponent es creado, automáticamente aparece en KoDeMat en la posición [0,0,0] con orientación [0,0,0] (el orden de las componentes es [x,z,y]). Como argumentos necesita, un nombre y la ruta en la que se encuentra el modelo en formato j3o.

- `public void addChildNode(long childId, long parentId)`

El método “addChildNode” establece una relación padre-hijo, en la que el hijo estará referenciado dentro de las coordenadas del padre.

- `public void setNodeTranslation(VisuComponent node, VisuVector3f nodePosition)`
- `public void setNodeRotation(VisuComponent node, VisuRotation nodePosition)`

Los métodos “setNodeTranslation” y “setNodeRotation” sirven para mover y orientar al componente respectivamente. Esta asignación de coordenadas se establece dentro de la referencia que tenga el componente, que bien puede ser la global o la del componente padre.

"VisuVector3f" y "VisuRotation" son tipos de datos propios de JMonkey. Para poder crear uno basta con utilizar el siguiente constructor:

- `new VisuVector3f(x, z, y);`

Éstas han sido las herramientas básicas con las que se han construido el resto de clases que se explican a continuación. En los anexos puede consultarse el código Java completo y comentado.

Clase HRL_API

La clase "HRL_API" es la principal y es la única clase utilizada por Matlab para comunicarse y enviar el estado de los sensores. Esta clase contiene toda lógica para la gestión de la visualización, utiliza las clases "HRL_Scenario" y "HRL_Component".

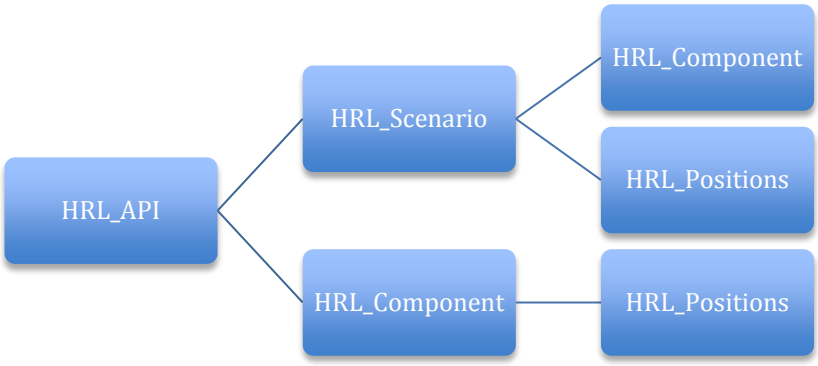

Fig. 8. Esquema de las relaciones existentes entre las clases Java desarrolladas

Para la gestión de la visualización se ha convertido el escenario en una especie de tablero de juego, en el que las diferentes casillas son las posibles posiciones, en las que una caja puede ser detectada por un sensor. Para tener un control acorde con este tablero, se han creado los siguientes tipos de datos: un HashMap, tanto para las posiciones detectables por los sensores como para las diferentes celdas del almacén y un HashMap para llevar un control de los nombres de las cajas.

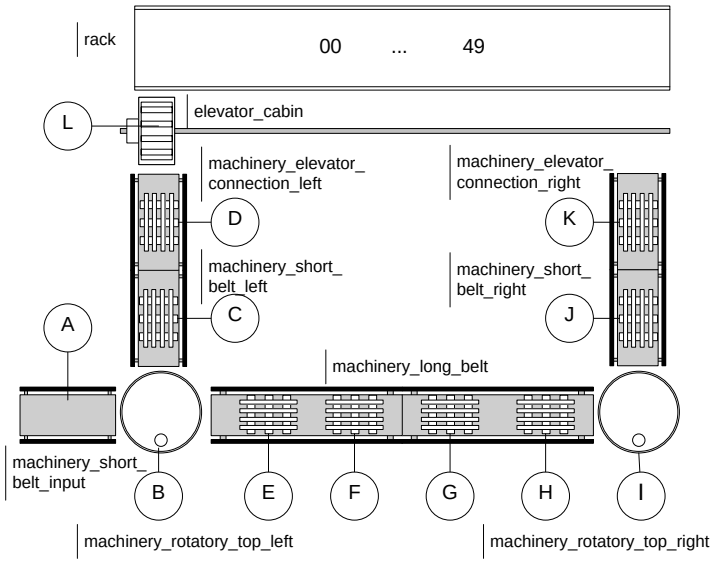

Fig. 9. Representación del almacén HRL en forma de tablero

En las siguientes tablas puede verse una representación del HashMap utilizado para las posibles posiciones, en realidad se han utilizado dos; uno para las posiciones en las máquinas designadas por letras, "sensorMap", y otro para las posiciones dentro del almacén, "rackMap".

- `public static HashMap <String, HRL_Component> sensorMap;`

A	B	C	D	E	F	G	H	I	J	K	L
Null	Box1	Null	Null	Null	Box2	Null	Box3	Null	Null	Null	Null

Table 2. Representación gráfica de la variable sensorMap

Cuando se detecte una caja en un sensor concreto, "HRL_API" creará un componente caja en la posición correspondiente del HashMap y la colocará en la posición homónima dentro del escenario virtual.

- `public static HashMap <Integer, HRL_Component> rackMap;`

00	01	02	03	...	...	...	...	46	47	48	49
Null	Null	Null	Null	...	...	...	...	Null	Box7	Null	Null

Table 3. Representación gráfica de la variable rackMap

"SensorMap" y "rackMap" son esencialmente el mismo tipo de mapa, la única diferencia es el tipo de dato utilizado como clave para acceder a su contenido. Ambas son HashMap que contienen componentes, esos componentes serán siempre cajas que se moverán por las diferentes posiciones del HashMap, acorde al avance de las cajas en el escenario real. La estantería cuenta con cincuenta celdas, la cifra de las decenas representa la fila de la estantería y la cifra de las unidades representa la columna.

Para el control de los nombres, se ha utilizado el HashMap "names", en el que se utiliza la propia clave de acceso a dato como nombre. El dato es una palabra que indica si ese número está libre ó no. A la hora de crear una nueva caja, se asignará como nombre el primer número de la lista que esté libre.

- `public static HashMap <Integer, String> names;`

1	2	3	4	5	6	7	8	...	...	...	...
busy	busy	free	busy	free	busy	free	free	...	...	...	...

Table 4. Representación gráfica de la variable names

"HRL_API" contará, además, con una variable de tipo "HRL_Scenario". La creación de esta variable supondrá la aparición de todos los componentes del escenario en el cliente de visualización.

- `public static HRL_Scenario scenario;`

Una vez conocidos los tipos de datos con los que la clase "HRL_API" funciona pueden explicarse los principales métodos. Variables y métodos han sido declarados cómo estáticos para que puedan ser llamados directamente desde Matlab sin la necesidad de crear previamente un objeto/instancia.

- `public static void start ()`

El método “start” debe ser llamado en primer lugar y una sóla vez. Su ejecución supone la creación de las variables de la clase y su inicialización. Los HashMaps “sensorMap” y “sensorRack” tendrán un estado inicial “null” mientras que “names” tendrá todas sus posiciones “free”. También se creará la variable “scenario” para que todos los componentes, maquinaria, elevador y almacén, aparezcan por primera vez en pantalla.

- `public static void updateSensorMap (boolean[] sensorMapMachines)`

El método "updateSensorMap" se encarga de la detección de los desplazamientos, aparición y eliminación de las cajas. Recibe como argumento un array de datos de tipo booleano enviado por Matlab y será utilizado para actualizar el estado de “sensorMap”. Cuando el algoritmo efectúa un cambio en el mapa, automáticamente se efectuará un cambio en la visualización; pudiendo observarse el movimiento, aparición, ó desaparición de una caja.

La distinción entre una aparición/desaparición de una caja y un movimiento se lleva a cabo comprobando la proximidad de las posibles posiciones dentro del tablero virtual representado por la variable “sensorMap”. Por ejemplo, si un sensor pasa a no detectar caja y uno de los sensores próximos a él detecta una, esto indica que se ha producido un desplazamiento y no una aparición/desaparición.

El modelo de almacén HRL no cuenta con sensores en las diferentes celdas de su estantería. La colocación de las cajas en las estanterías se consigue mediante la detección de la aparición ó desaparición de una caja en la cabina del elevador. Es decir, si el elevador se desplaza a una determinada posición con una caja y ésta desaparece, el sistema colocará la caja en la celda correspondiente de la estantería; o bien, si el elevador está en una determinada posición del almacén y se detecta un objeto en la cabina, ello implicará que se ha extraído una caja del almacén, por tanto, este método también es el encargado de actualizar el estado de “rackMap".

- `public static void updateScenario (String tool, int rotatoryLeft, int rotatoryRight)`

El método "updateScenario" actualizará el estado de los elementos móviles del escenario a excepción del elevador, estos elementos son las mesas giratorias y la máquina herramienta. La orientación de las mesas es enviada por Matlab en forma de número entero y ha de ser interpretado como grados, los datos serán recogidos por las variables “rotatoryLeft” y “rotatoryRight”. La máquina herramienta, en cambio, tiene dos posibles estados, arriba y abajo, y será almacenado en la variable “tool”.

- `public static void updateElevator (boolean[] sensorMapElevatorCols, boolean[] sensorMapElevatorRows)`

El método "updateElevator" se encarga, exclusivamente, de mover el elevador en el escenario 3D. Recibe dos arrays: sensorMapElevatorCols y sensorMapElevatorRows. El primero de ellos está compuesto por diez elementos booleanos que representan las diez columnas de la estantería entre las que puede moverse la máquina. El segundo está compuesto por seis elementos, los cinco primeros corresponden con las diferentes filas del almacén y el último representa la altura a la que están las máquinas y que será la altura

a la que una caja pasa de las cintas transportadoras a la cabina elevadora. Es obvio que solamente un dato de cada array podrá ser "true" en un mismo instante.

- `public void rackInitialState (boolean[] rack)`

Cómo se ha mencionado, el modelo de almacén HRL no cuenta con sensores en las diferentes celdas. Una forma de detectar si hay cajas en la estantería en el momento en el que se inicia el sistema es que el sistema SAP envíe la información a través de ésta función, rackInitialState.

Para un conocimiento del código más en detalle es necesario consultar los anexos del documento, allí se encuentra el código completo y comentado.

Clase HRL_Component

El objetivo de esta clase es representar un componente cualquiera del HRL (almacén de gran altura). Básicamente, este tipo de objeto utiliza métodos muy similares a los descritos aguas arriba, más algunos específicos para el modelo de almacén designado como HRL en el que está basado el proyecto. Las variables que la clase utiliza son las siguientes:

- `public VisuComponent model;`

La variable "model" de tipo VisuComponent, tipo de dato que pertenece al cliente de visualización. Cuando un VisuComponent es creado automáticamente aparece en la ventana de KoDeMat.

- `public HRL_Positions positions;`

La clase HRL_Component deberá crear, además, un objeto "HRL_Positions". Este objeto será utilizado por algunos de los métodos de la clase para poder mover su objeto VisuComponent a posiciones concretas.

- `public HRL_Component (String name, String designator, float x, float z, float y)`

"HRL_Component" es el constructor de la clase. Necesita un nombre "name" para el objeto y el nombre del modelo 3D "designator". Requiere de unas coordenadas para colocar el componente en la escena.

- `public HRL_Component (String name, String designator)`

Esta es otra versión de constructor de objeto, no requiere de coordenadas y colocará el componente en la posición por defecto [0,0,0].

- `public void delete ()`

El método "delete" eliminará el componente de la escena. No basta con eliminar el objeto HRL_Component para que este desaparezca del cliente de visualización.

- `public void setOrientation (float x, float z, float y)`

El método "setOrientation" establece una orientación en el objeto dentro del sistema de referencia en el que se encuentre, podrá ser el absoluto o el de un componente padre.

- `public void setPosition (float x, float z, float y)`

El método "setPosition" coloca el objeto en una posición concreta dentro del sistema de referencia en el que se encuentre, podrá ser el absoluto o el de un componente padre.

A continuación, los siguientes métodos "setPositionIn" colocan el componente frente al sensor del componente máquina "component", que es recibido como argumento.

- `public void setPositionIn (HRL_Component component, int sensor)`

Esta versión colocará el componente dentro del la maquina transportadora "machinery_long_belt" frente a uno de los cuatro sensores de los que dispone, argumento "sensor". Sólo admitirá como argumento la máquina "machinery_long_belt".

- `public void setPositionIn (HRL_Component component)`

Esta versión sirve para colocar el componente en cualquier máquina que solo cuente con un sensor. Estas son todas excepto "machinery_long_belt".

- `public void setPositionIn (HRL_Component component, int row, int col)`

Este versión se encarga de colocar al componente dentro del almacén. Solo admitirá como argumento "component" el componente "rack" y deberá introducirse en los argumentos "row" y "col" la fila y columna de la celda en la que se quiere colocar al componente, desde 0 a 4 y 0 a 9 respectivamente.

- `public void childOf (HRL_Component father)`

El método "childOf" cambia el sistema de referencia en el que se encuentra el componente por el sistema de referencia del componente padre, argumento "father".

Clase HRL_Scenario

La clase HRL_Scenario representa el conjunto de componentes del escenario a excepción del componente caja. En su definición se encuentran tantos objetos de tipo HRL_Component como componentes tiene el escenario. La lista es la siguiente:

- `public HRL_Component floor;`
- `public HRL_Component rack;`
- `public HRL_Component railway;`
- `public HRL_Component elevator_tower;`
- `public HRL_Component elevator_cabin;`
- `public HRL_Component machinery_elevator_connection_left;`
- `public HRL_Component machinery_elevator_connection_right;`
- `public HRL_Component machinery_short_belt_left;`
- `public HRL_Component machinery_short_belt_right;`
- `public HRL_Component machinery_rotatory_bottom_left;`
- `public HRL_Component machinery_rotatory_bottom_right;`

- `public HRL_Component machinery_rotatory_top_left;`
- `public HRL_Component machinery_rotatory_top_right;`
- `public HRL_Component machinery_long_belt;`
- `public HRL_Component machinery_short_belt_input;`
- `public HRL_Component machinery_tool_machine_tower;`
- `public HRL_Component machinery_tool_machine_tool;`

Cuando un objeto de esta clase se crea, su constructor iniciará todas las variables definidas y aparecerán automáticamente en el cliente de visualización KoDeMat. Para su colocación en posiciones concretas dentro del escenario la clase también necesitará incluir un objeto HRL_Positions. Los métodos principales de HRL_Scenario se describen a continuación:

- `public HRL_Scenario ()`

"HRL_Scenario" es el constructor de la clase. Se encarga de crear todos los objetos HRL_Component declarados y de establecer sus relaciones padre-hijo.

- `public HRL_Component getHRL_ScenarioComponet(String key)`

El método "getHRL_ScenarioComponent" devuelve uno de los objetos HRL_Component del escenario según la nomenclatura establecida para cada una de las posibles posiciones en el escenario. Ver Fig. 9. Representación del almacén HRL en forma de tablero.

- `public void toolUp()`
- `public void toolDown()`

Los métodos "toolUp" y "toolDown" se encargan de colocar arriba y abajo la máquina herramienta respectivamente.

- `public void rotatoryLeft (int degrees)`
- `public void rotatoryRight (int degrees)`

Los métodos "rotatoryLeft" y "rotatoryRight" se encargan de orientar las mesas giratorias colocadas en el lado izquierdo y derecho del escenario respectivamente. Su argumento es la orientación respecto al eje Z dada en grados.

- `public void moveElevator(int row, int col)`

El método "moveElevator" se encarga de mover el elevador a una de las sesenta posiciones posibles, cincuenta son las celdas del almacén, las diez restantes han de entenderse como una fila extra para representar la altura a la que el elevador debe recoger ó depositar una caja en las cintas transportadoras.

Clase HRL_Positions

La clase HRL_Positions ha sido creada para almacenar todas las coordenadas en las que se encuentran los componentes en el almacén. Se trata de una librería de datos y las clases HRL_Scenario y HRL_Component deben incluir su definición para poder acceder a esas coordenadas. Esta clase está compuesta únicamente de constantes y por tanto, no requiere mayor atención. Puede consultarse en los anexos del documento junto con las otras clases.

4.4 El control desde Simulink

Las llamadas a los métodos descritos en el apartado anterior se realizan automáticamente y constantemente desde una función en Simulink. La función sólo dispone de entradas, que son la lectura de los sensores y de los servomotores de las mesas giratorias. La lectura de los servomotores servirá para definir la orientación en la que se encuentran las mesas.

Cuando la aplicación Simulink comienza a funcionar, la primera operación es la llamada al método "start". Esta llamada se realizará una única vez, ya que sirve para inicializar las variables que utiliza la API de visualización desarrollada, HRL_API. A continuación, el cliente de visualización creará automáticamente el escenario.

- `javaMethod('start','kodemat.hrl.client.HRL_API');`

Una vez que la función ha inicializado el sistema, pasará a invocar continuamente a los métodos de la HRL_API encargados de la gestión de la visualización.

- `javaMethod('updateSensorMap','kodemat.hrl.client.HRL_API',[A,B,C,D,E,F,G,H,I,J,K,L]);`
- `javaMethod('updateElevator', 'kodemat.hrl.client.HRL_API', [C0, C1,C2,C3,C4,C5,C6,C7,C8,C9], [R0,R1,R2,R3,R4,R5]);`
- `javaMethod('updateScenario', 'kodemat.hrl.client.HRL_API', tool, rotatoryLeft, rotatoryRight);`

En color violeta y entre comillas simples se encuentran respectivamente, el nombre del método y la ruta en la que se encuentra dicho método. Todos los métodos pertenecen a la clase HRL_API que se encuentra en: `'kodemat.hrl.client.HRL_API'`. La ruta tiene su localización dentro del archivo jar, que se ha añadido previamente a Matlab. Los argumentos que se envían son las variables de entrada a la función y representan el estado de los sensores en el sistema de almacenamiento.

Para enviar un array a un método Java es necesario hacerlo directamente en la propia llamada, no es posible enviar un array por referencia. Esto es enviando directamente los valores colocándolos entre corchetes y separados por comas.

4.5 Simulación del sistema

Para simular el comportamiento del escenario se han utilizado dos técnicas: una manual en la que es posible un análisis más exhaustivo gracias a su interactividad y otra automática, en la que es posible plantear un circuito ejemplo y proceder a su observación.

Simulación manual

En este tipo de simulación, se han utilizado interruptores que conmutan entre 1 y 0 (en Matlab equivalen, a la vez, a estados true y false respectivamente). Así, es posible comprobar el correcto funcionamiento de todo el sistema de visualización, provocando la aparición, desaparición y desplazamientos de las cajas en cualquier punto del escenario.

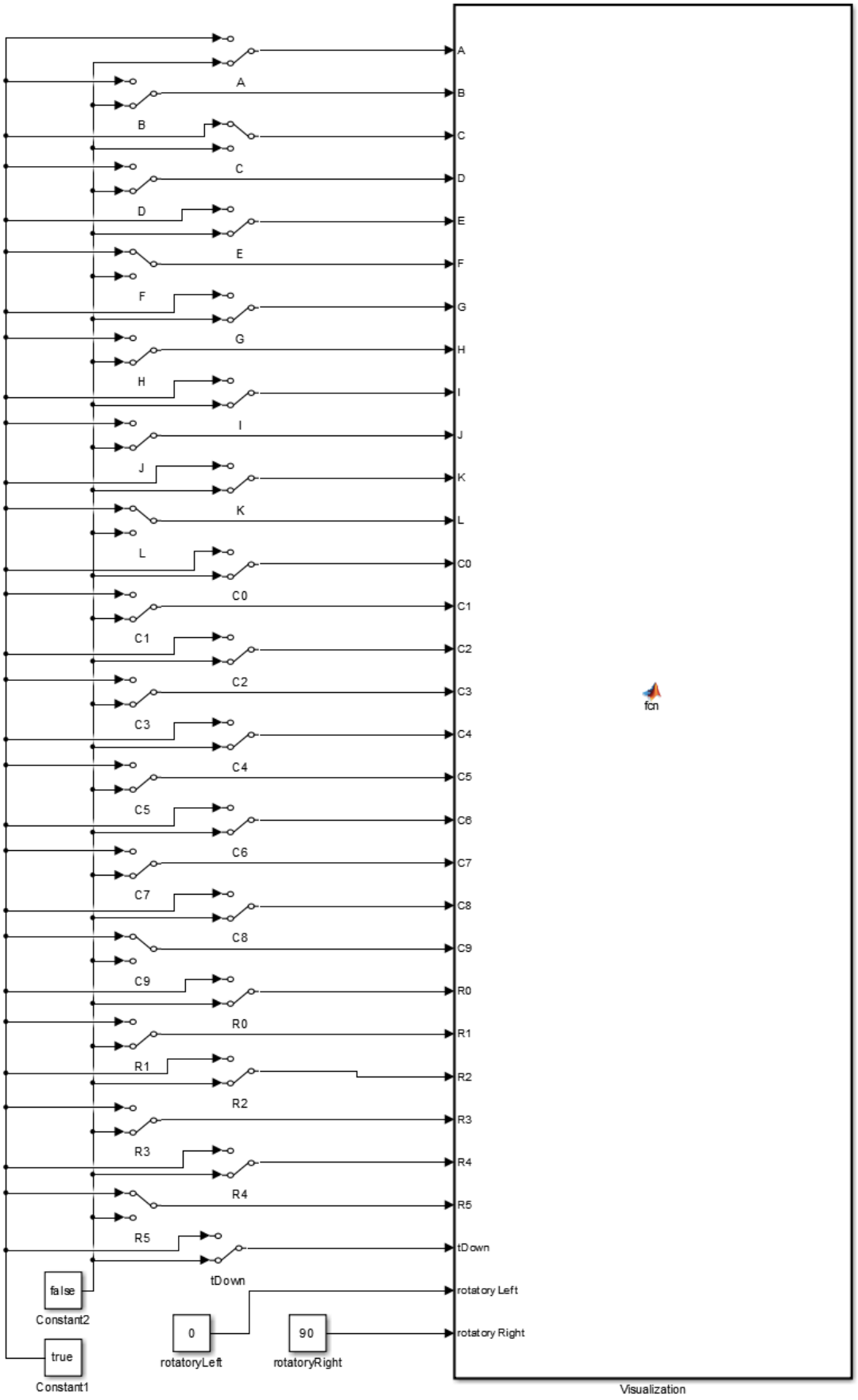

Fig. 10. Función Visualization, simulación manual

En la Fig. 10. Función Visualization, simulación manual, se observa la función "Visualization". El algoritmo de esta función se ha comentado en el apartado anterior y puede consultarse de forma completa en el apartado anexos.

La función está conectada a diferentes interruptores. Los interruptores A-L corresponden a los diferentes puntos del escenario, entre los que una caja puede moverse y representan el estado de los sensores. Si su estado es "on" implicará que hay una caja en frente del sensor y viceversa.

Los interruptores "Cn" y "Rn" corresponden a los sensores con los que cuenta el elevador. En la visualización, la máquina podrá moverse entre sesenta posiciones posibles, cincuenta corresponden a las celdas del almacén y las diez restantes son una fila extra que simboliza la altura de conexión entre el elevador y las cintas transportadoras.

El interruptor "tDown" coincide con el sensor de la máquina herramienta en posición de taladrado. Si "tDown" está en estado "on" la herramienta bajará a dicha posición, de lo contrario permanecerá arriba, en posición de reposo.

Con "rotatoryLeft" y "rotatoryRight" se establece la orientación de las mesas giratorias. Representan la lectura de posición en la que se encuentran los servomotores que controlan estas mesas.

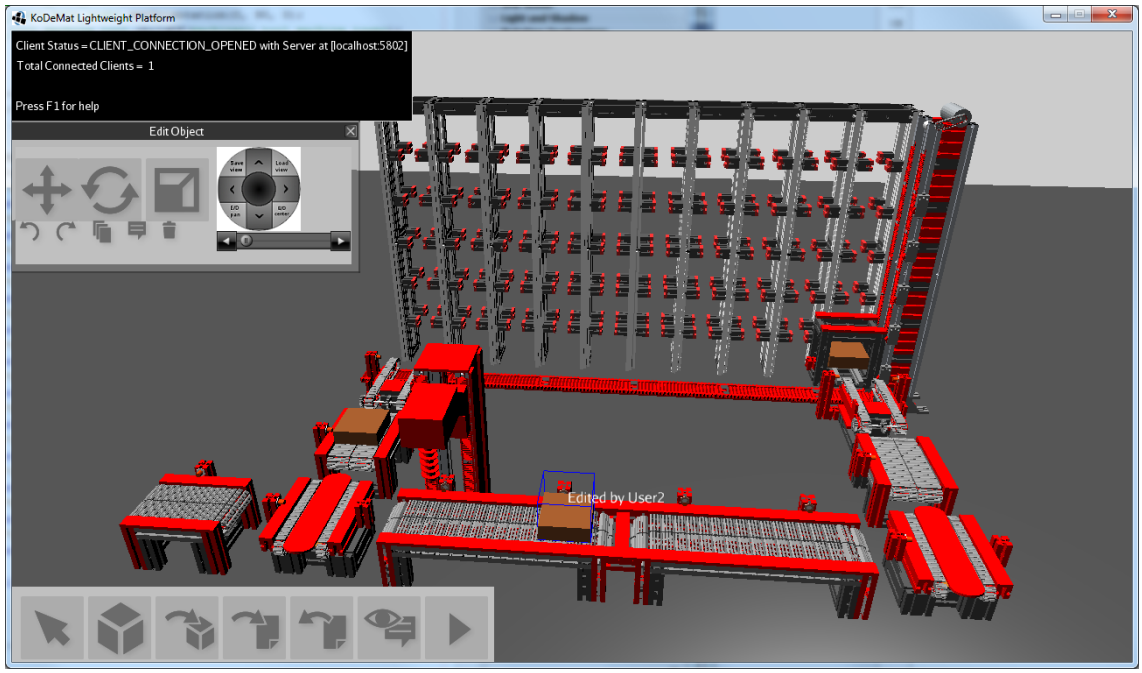

Fig. 11. Visualización 3D del sistema de almacenamiento y recuperación en KoDeMat

Simulación automática

Este nuevo apartado trata de cómo simular el sistema completo y observar la visualización de forma automática. En otras palabras, en Simulink se han modelado las distintas máquinas y parte de la lógica que llevaría el PLC que controla el almacén real. El objetivo es implementar un pequeño recorrido para una o varias cajas y comprobar cómo el sistema de visualización es capaz de representarlo.

En la Fig. 12. Esquema en Simulink para la simulación automática de la visualización, puede verse el esquema Simulink para tal efecto, en el que se ha utilizado una versión reducida de la función "Visualization".

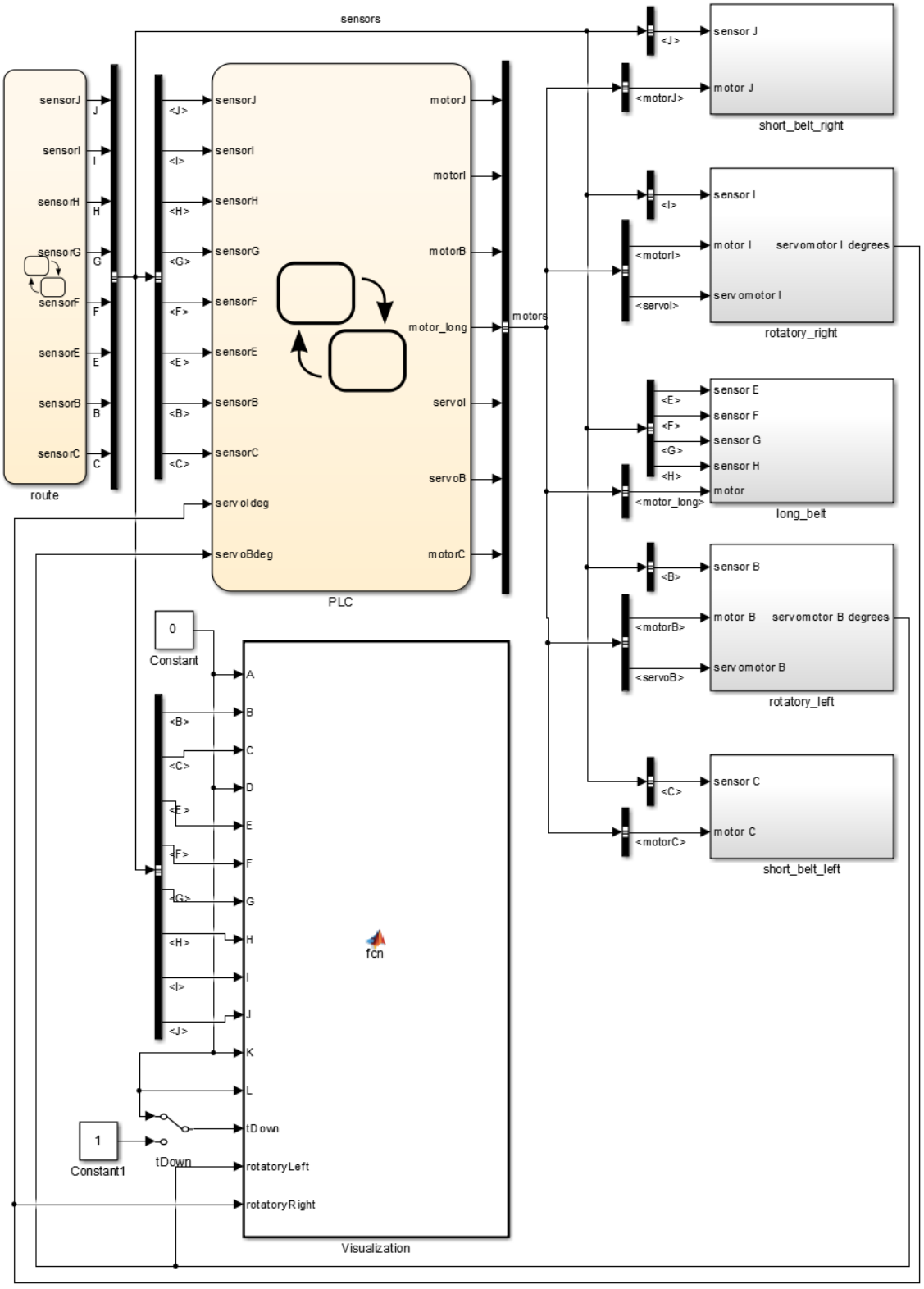

Fig. 12. Esquema en Simulink para la simulación automática de la visualización

El bloque "route" es el encargado de simular un circuito. Conforme avanza el tiempo, el bloque activa y desactiva sus salidas, representando el estado de los sensores.

Las diferentes máquinas del sistema de almacenamiento y recuperación están representadas por los bloques en color gris, situados a la derecha de la imagen. En su interior se encuentran unos displays para mostrar el estado de su sensor ó sensores y de su motor ó motores. En el caso particular de las mesas giratorias, se cuenta con un bloque contador, que simula la evolución de la posición de los servomotores.

El bloque "PLC", como su nombre indica, simula una pequeña parte de la lógica del PLC real, lee el estado de los sensores gobernados por el bloque "route" y controla los motores de las máquinas. El bloque también recibe la realimentación de la posición de los servomotores para conocer en todo momento la orientación de la máquina y actuar según corresponda.

Por último, situada en la parte inferior, se encuentra la función "Visualization", que lee el valor de las variables del sistema y envía esta información al cliente de visualización.

5 Trabajo futuro

Tras la verificación de un correcto funcionamiento del sistema de visualización, en la etapa de simulación, llegaría el momento de proceder con pruebas reales. La comunicación entre la maqueta y el ordenador utilizado como unidad de control se realiza con mensajes SOAP, por lo que resulta posible acceder a la información que el PLC envía y capturar desde Simulink, el estado de los sensores que sean de interés para la visualización.

Con el modelo acoplado al cliente de visualización y en correcto funcionamiento, podría plantearse la posibilidad de mejorar las secuencias de animación. Por ejemplo, que el sistema fuese capaz de detectar la velocidad de los motores, ó que el desplazamiento de las cajas de un punto a otro se realice a la misma velocidad que en el modelo real.

En última instancia, se incorporaría la tecnología RFID (Radio Frequency IDentification) para poder identificar unívocamente los pedidos.

6 Conclusiones y agradecimientos

He de decir que este proyecto comenzó, desde un punto de vista personal, dos meses antes de viajar a Alemania. Durante esos dos meses adquirí unos conocimientos básicos de programación Java a través de "Udacity", un sitio web de autoaprendizaje a través de video-lecciones y la realización de ejercicios de forma interactiva.

El desarrollo de este TFG me ha dado la oportunidad de trabajar en el campo de la innovación logística y he podido colaborar en investigaciones llevadas a cabo por la Universidad Técnica de Múnich. Gracias a este TFG he aprendido y manejado conceptos y herramientas nuevos, tanto en el ámbito de software como en el de programación orientada a objetos.

La realización del TFG fuera de España no sólo ha supuesto un reto para mí desde un punto de vista académico, es también un reto personal. Vivir fuera es siempre complicado y supone un gran paso adelante a la hora de crecer como persona y como ingeniero.

Me gustaría mencionar a la asociación IAESTE, International Association for the Exchange of Students for Technical Experience, ya que gracias a ella he podido conseguir esta beca de colaboración en la Universidad de Múnich. Por ello, he de dar las gracias y felicitar a todos los estudiantes que colaboran, no solo de Zaragoza y Múnich sino de todo el mundo, por todo el trabajo que hacen para que esta asociación funcione.